**KUBERNETES**

* Kubernetes is an open-source container management tool that Automates Container Deployment, Container Scaling, and Load Balancing.
* It schedules, runs and manages isolated containers that are running on Virtual/Physical/cloud machines.
* It Is supported by all the Cloud providers.

**HISTORY**

* Google developed an internal system called “BROG” (later names OMEGA) to deploy and manages Thousands of Google Application and services and their cluster.
* In 2014, google introduced Kubernetes as an open-source platform.
* Kubernetes is written in Go-lang.
* Kubernetes was later donated to Cloud Native Computing Foundation (CNCF).
* Cloud-native means it will develop existing features in the cloud for better purpose usage.

**ONLINE PLATFROMS for K8s**

* Kubernetes playground.
* Kubernetes with K8s
* Kubernetes with Kubernetes classroom.

**CLOUD-BASED K8s SERVICES**

* Google Kubernetes services.
* Azure Kubernetes services.
* Elastic Kubernetes services.

**INSTALLATION TOOLS**

* Mini-kube (single node)
* Kube-adm (Multi node)
* Kops (Multi node & Infra)

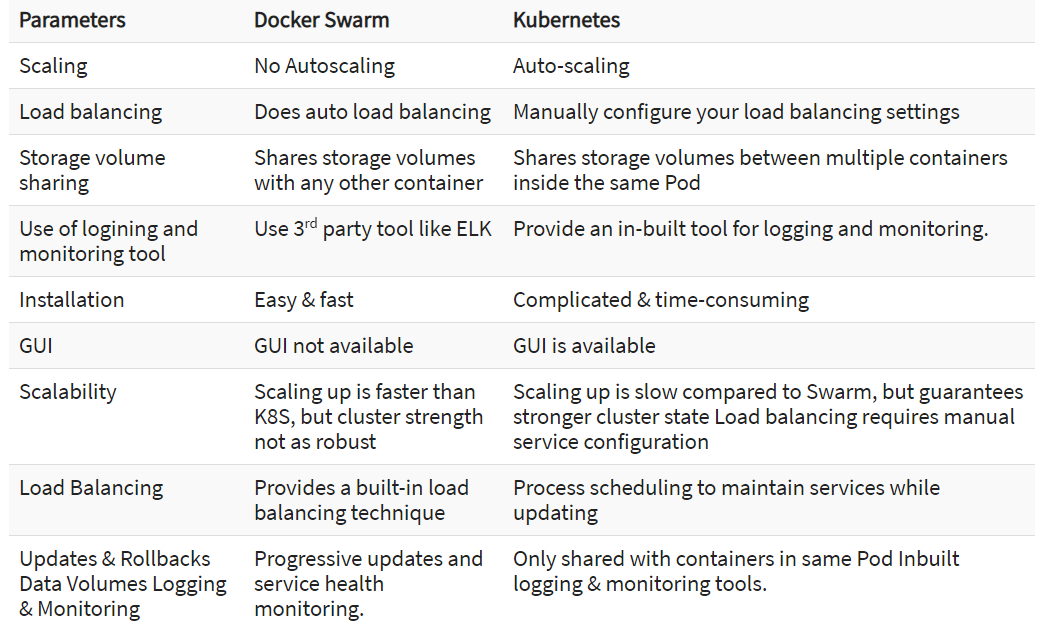
**CONTAINER SCALEUP PROBLEMS**

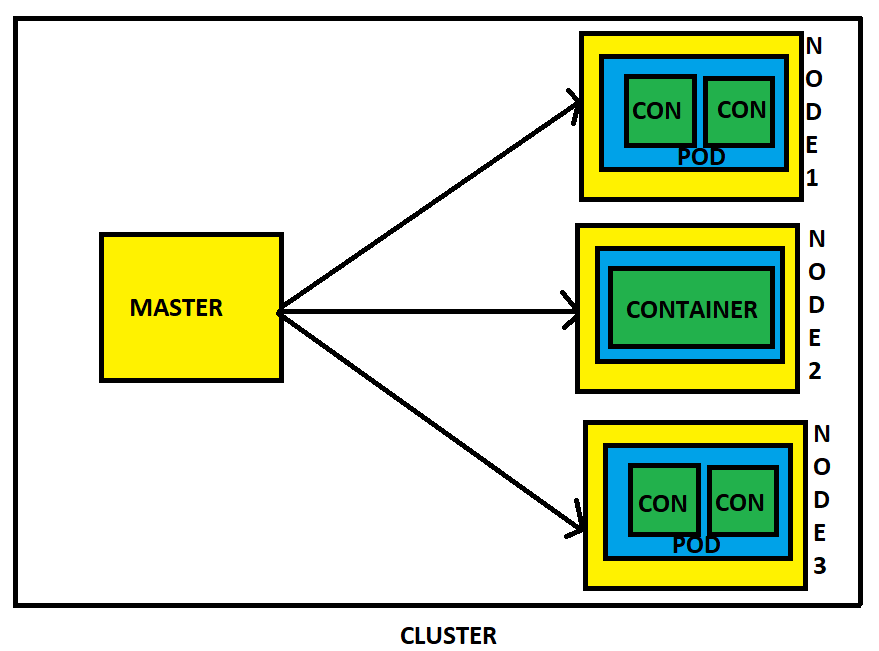
* Containers cannot communicate with each other.
* Auto-scaling and Load balancing were not possible.
* Containers had to be managed carefully.

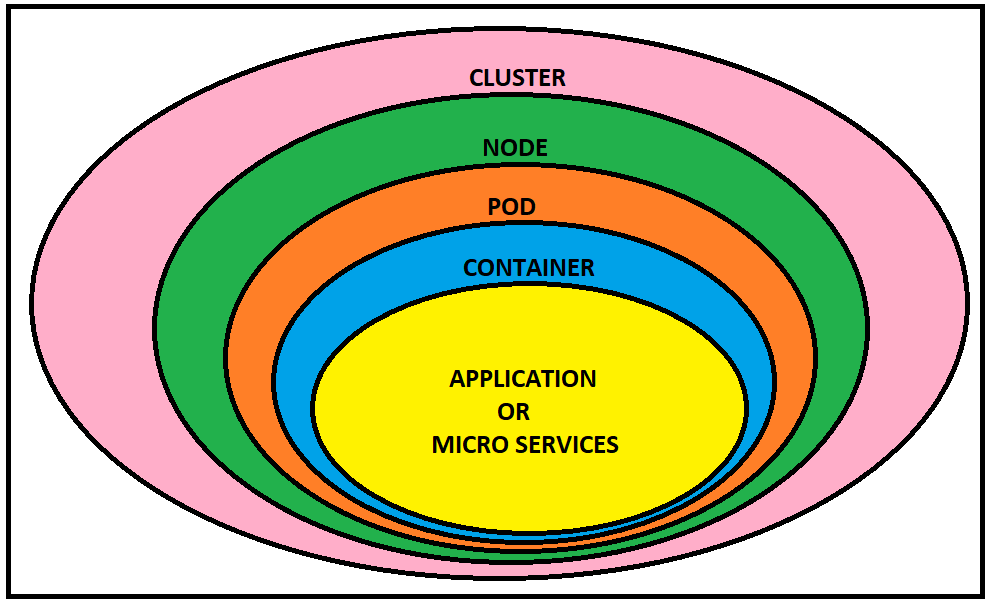
**FEATURES**

* Orchestration (Clustering of any number of containers running on different networks).
* Auto-scaling (Vertical [Existing] - >most Preferable and Horizontal [New]) and Auto-healing.
* Load balancing.
* Platform Independent (Cloud/Virtual/Physical).
* Fault tolerance (Node/Pod failure).
* Roll back (Going back to previous version).
* Health Monitoring of Containers. If one Container fails it will create another container.
* Batch Execution (One time, Sequential, Parallel).
* Scripts in K8s is called Manifest, which is in form of JSON or YAML.

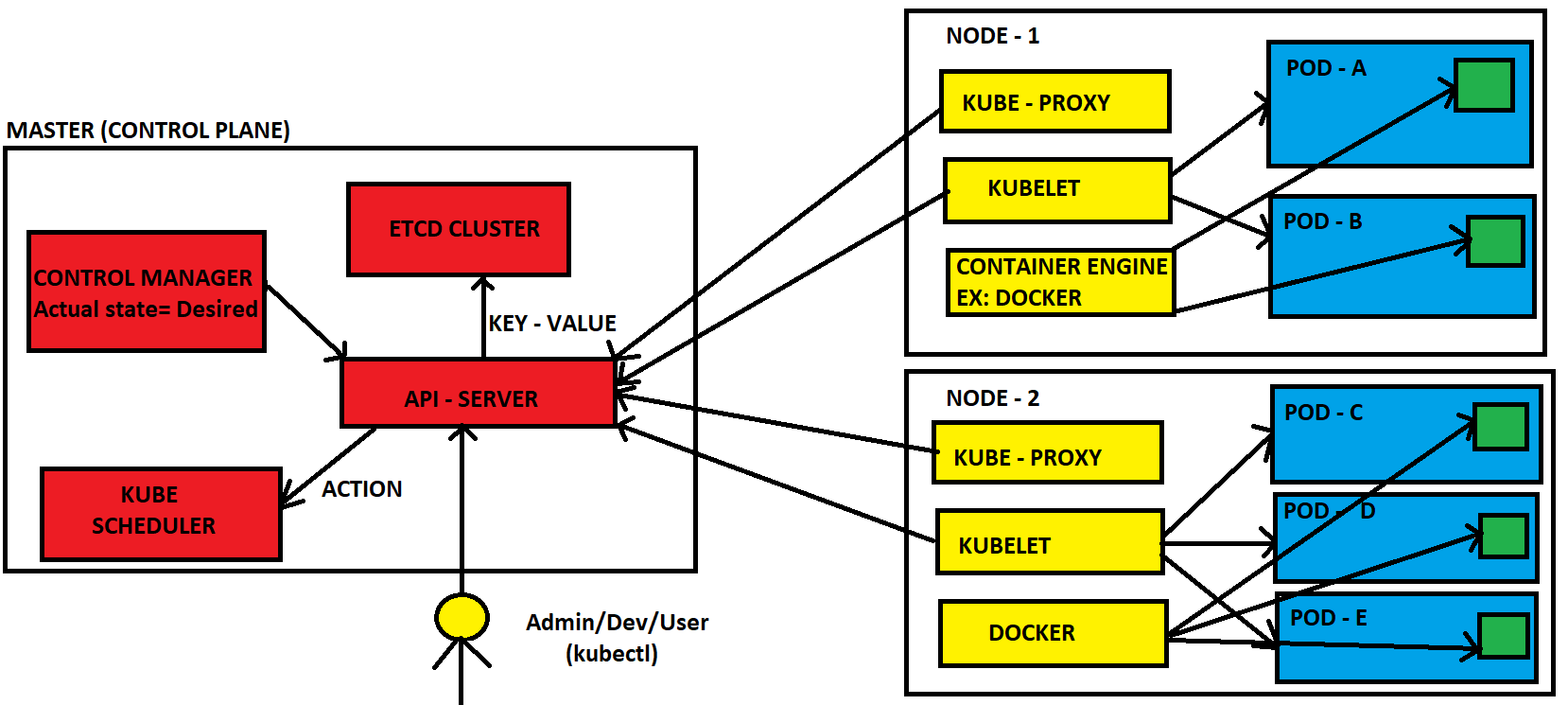
**DOCKER SWARM VS K8s**







**ARCHITECTURE**

**Master components**

* **API Server:** Exposes the API.
* **ETCD Cluster:** Key-value stores all cluster data. (Can be run on the same server as a master node or on a dedicated cluster.) Check if the pod is present or not in worker. If present then it will go to the kube scheduler.
* **Kube-scheduler:** Schedules new pods on worker nodes.
* **Kube-controller-manager:** Runs the controllers. And check whether the pod is running or not for every 5 sec.
* **Cloud-controller-manager:** Talks to cloud providers.

**Node components**

* **Kube-proxy:** Keeps network rules like Addressing IP to Pods.
* **Kubelet: It is the ain component in worker node** Agent that ensures containers in a pod are running.
* **Container engine:** Maintains the containers like Docker, Rocket etc.

**Pod:** refers one or more containers that should be controlled by a single application.

**Service:** An abstraction that defines a logical set of pods as well as the policy for accessing them.

**Volume:** It is similar like a docker volume. A k8’s volume applies to the whole pod and mounted on all contaiiners in the pod.

**Namespace:** A segment of the cluster dedicated to a certain purpose. Namespace must be unique and cannot access the resources in different name spaces.

**Replica-Set (RS):** Ensures that desired amount of pod is what’s running.

**Deployment:** Describes the desired state of the pod in YAML file.

**Stateful-Set:** A workload API object that manages stateful applications, such as databases.

**Daemon-Set:** Ensures that all or some worker nodes run a copy of a pod. This is useful for daemon applications like fluentd.

**Job:** Creates one or more pods, runs a certain task(s) to completion, then deletes the pod(s).

**WORKING WITH K8s**

* We create Manifest (.yml) format.
* Apply this to the Cluster (to Master) to bring into the desired state.
* Pod run on the node, which is controlled by the master.

**ROLE OF MASTER**

* K8s Cluster contains Running containers or Physical/VM/cloud instance/all mix.
* It designates on or more of these as master and all others as Workers or Nodes.
* Master is going to run set of all K8s processes. These processes will ensure smooth functioning of cluster. This process is called as “control plane”.
* Can be multi-master for high Availability.
* Master runs control plane to run the cluster smoothly.

**COMPONENTS OF CONTROL PLANE**

**KUBE-API SERVER** (For all Communications)

* It is front end of the Control Plane.
* It will directly interact with the user (I.e we apply .yml or json manifest to kube-Apiserver).
* This kube-Apiserver is directly meant to scale Automatically as per load.

**ETDC (Database)**

* It is nothing but the Data base which Stores Meta data and status of cluster.
* It is consistent and high available store (Key - Value store).
* Source of touch for Cluster state (information about the state of Cluster).

**FEATURES**

* **Fully replicated**: Entire state is available on every node in the cluster.
* **Secure**: Implements Automatic TLS with operational Client-Certificate Authentication.
* **Fast**: Benchmarked t 10,000 writes per second.

**KUBE-SCHEDULER (schedule pods)**

* When user makes a request for creation and management of pods, Kube-scheduler is going to take the action on those requests, Handles pod creation and Management.
* Kube-scheduler match/assign any node to create and run pod.
* A scheduler watches for newly created pods that have no node assigned for every pod that the scheduler discovers, it becomes responsible for finding best node for that pod to run on.
* It gets the information for hardware configuration from configuration files and schedules the pods on the nodes Accordingly.

**CONTROL MANAGER**

* Makes sure actual state of cluster matches to the desired state.

Two possible choices for Control manager

1. If K8s on cloud, then it will be Cloud-Controller-Manager.
2. If K8s on non-cloud, then it will be Kube-Controller-Manager.

Components on master that runs Controller.

* **Node-controller**: For checking the cloud provider to determine if a node has been detected in the cloud after it stops responding.
* **Route-controller**: For setting up network, routes on your cloud.
* **Service-controller**: For load balancers on your cloud against services of type Load Balancers.
* **Volume-controller**: For Creating, attaching and mounting volumes and interacting with the cloud provider to Orchestrate Volume.

**NODE COMPONENTS**

**KUBELET**

* It Is nothing but Agent running on the Node.
* Listens to the Kubernetes Master (Ex: Pod creation request).
* Uses the port :10255 (can be changeable).
* Sends Successful/Fail reports to the Master.

**CONTAINER ENGINE**

* It can be Docker, Rocket or any other.
* Works with Kubelet.
* Pulling images.
* Start/Stop the Containers.
* Exposing Containers on ports specified in the Manifest.

**KUBE-PROXY**

* Assigns IP to each pod.
* It is required to assign IP addresses to Pods (Dynamic IP).
* It runs on each node & this will make sure that each pod will gets its unique IP address.

All the three components Kubelet, Kube-Proxy and Container engine Is collectively called as Node.

**POD**

* Smallest unit of the Kubernetes.
* It is a group of one or more containers that are deployed together on the same Host.
* A cluster is a group of nodes.
* A cluster has atleast one master node and one worker node.
* In Kubernetes, control unit is the pod, not containers.
* Consistent of one or more Tightly coupled containers.
* Pod runs on node, which is controlled by master.
* Kubernetes only knows about Pods (it does not know about individual container).
* Cannot start container without a Pod.
* One Pod usually contain only one container.

**MULTI CONTAINER PODS**

* Share access to memory space.
* Connect to each other using localhost. <container port>
* Share access to the same Volume.
* Containers with in the pods are deployed in an all-or-nothing manner.
* Entire pod is hosted on the same node (scheduler will decide about which node).

**LIMITATIONS**

* By default, no Auto-scaling or Auto-scaling, we need to do it Manually.
* POD Crashes, Solution for that is Higher level K8s Objects but, we need to add these.

**Higher level K8s Objects**

* **Replication-set**: Scaling and Healing.
* **Deployment**: Versioning and Roll back.
* **Service**: Static (Non-Ephemeral) IP and Networking.
* **Volume**: Non-Ephemeral Storage.

**IMPORTANT NOTATIONS**

* Kubectl : Single Cloud.
* Kubeadm : On-Premises.
* Kubefed : Federated (Hybrid cloud).

**MINI KUBE:**

**Minikube**creates a single node cluster inside a VM or Cloud Instance. It is good for beginners to

learn Kubernetes since you don’t have to create a master and worker node to create a cluster and

we can practice basic Kubernetes functions and can also install the Kubernetes dashboard on it.

**PRE-REQUISTES**

* Minimum 2 CPU’s or more
* Minimum 2GB of free memory
* Minimum 20GB of free disk space
* Internet connection
* Container or virtual machine manager, such as: Docker, Hyperkit, Hyper-V, KVM, Parallels, Podman, VirtualBox, or VMware Fusion/Workstation

**KUBECTL INSTALLATION:**

* sudo apt update -y
* curl -LO https://storage.googleapis.com/kubernetes-release/release/`curl -s https://storage.googleapis.com/kubernetes-release/release/stable.txt`/bin/linux/amd64/kubectl
* chmod +x ./kubectl
* sudo mv ./kubectl /usr/local/bin/kubectl
* kubectl version

**DOCKER INSTALLATION:**

* sudo apt-get install docker.io -y
* sudo systemctl status docker
* sudo usermod -aG docker $USER && newgrp docker

**MINIKUBE INSTALLATION:**

* curl -Lo minikube <https://storage.googleapis.com/minikube/releases/latest/minikube-linux-amd64>
* chmod +x minikube
* sudo mv minikube /usr/local/bin/
* minikube version
* sudo minikube start --vm-driver=none
* X Exiting due to GUEST\_MISSING\_CONNTRACK: Sorry, Kubernetes 1.22.3 requires conntrack to be installed in root's path
* sudo apt-get install -y conntrack
* minikube start --vm-driver=none
* minikube status
* kubectl cluster-info
* kubectl get events
* kubectl config view
* kubectl run hello-minikube --image=gcr.io/google\_containers/echoserver:1.4 --port=8080
* kubectl get pods

**DEPLOYING AN APP**

* kubectl create deployment hello-node --image=k8s.gcr.io/echoserver:1.4
* kubectl get deployment
* kubectl expose deployment hello-node --type=NodePort --port=8080
* kubectl get svc
* curl -v public-ip:32548
* minikube ip
* curl -v private-ip:32548
* <http://private-ip:32548/>
* kubectl delete service hello-node
* kubectl delete deployment hello-node
* minikube stop
* minikube delete

**KUBECTL:**

* It is the Kubernetes command-line tool.
* It can communicate with a Kubernetes cluster's [control plane](https://kubernetes.io/docs/reference/glossary/?all=true#term-control-plane), using the Kubernetes API.
* It allows you to run commands against Kubernetes clusters.
* You can use kubectl to deploy applications, inspect and manage cluster resources, and view logs.
* The configuration file is located on .kube directory.

SYNTAX:

kubectl [command] [TYPE] [NAME] [flags]

Command: Operation you want to do (create, delete, get, describe)

TYPE: Specifies the resource type, it Is case sensitive.

NAME: Specifies the resource Name, it Is case sensitive.

REPLICASET:

it is nothing but Group of pods

if one pod is delete, terminated or failed then it will create another one.

It will maintain a stable set of replica Pods running at any given time.

It is the enhance version of Replication controller.

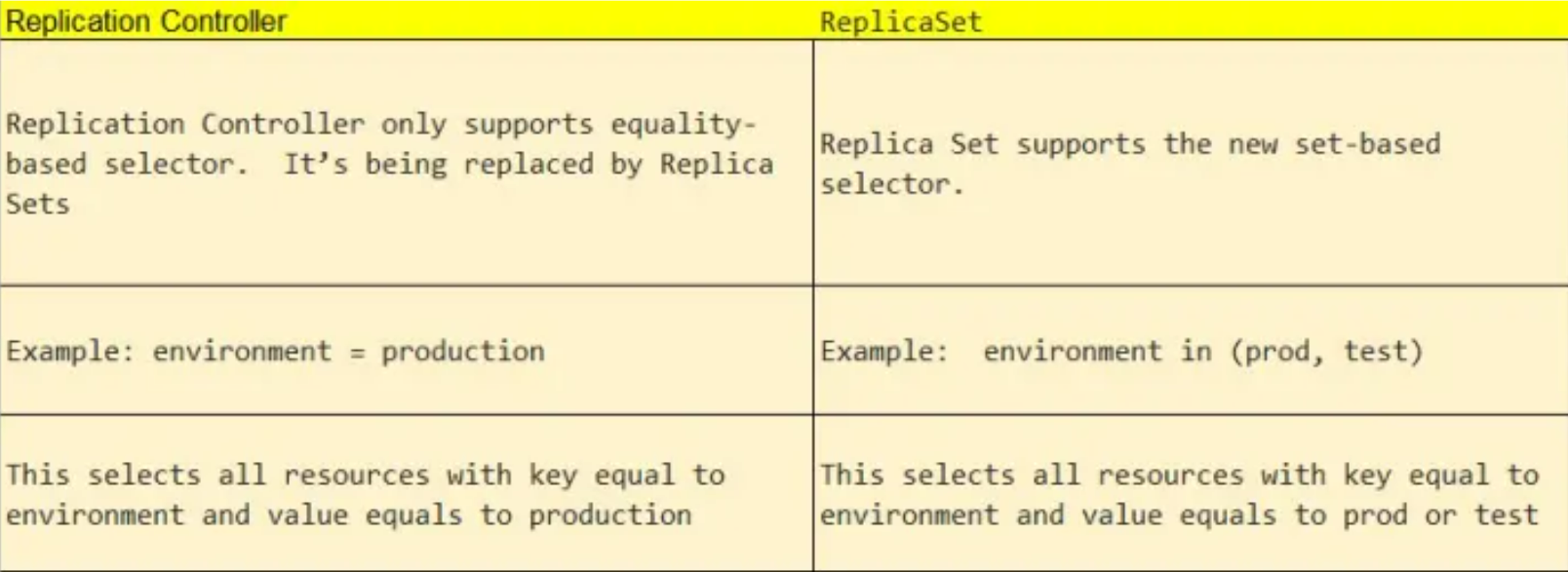
We can create multiple pods to share the load among them by using the replicaset.

It can balance the load even the pod is on different node.

If number of users increasing we can increate replica sets.

## Replication Controller

Replication Controller is one of the key features of Kubernetes, which is responsible for managing the pod lifecycle. It is responsible for ensuring that the specified number of pod replicas are running at any point in time. It is used in times when one wants to make sure that the specified number of pods or at least one pod is running. It has the capability to bring up or down the specified no of pods.



KIND:

POD, SERVICE -- > V1 & REPLICASET, DEPLOYMENT -- > apps/v1

WORKING:

It has features of Replicaset and some other extra features like updating and rollbacking to a particular version we want without downtime\*.

here application will reside on pod to manage and update& manage that pod we used deployment

A Deployment provides declarative updates for [Pods](https://kubernetes.io/docs/concepts/workloads/pods/) and [ReplicaSets](https://kubernetes.io/docs/concepts/workloads/controllers/replicaset/" \t "_blank).

You describe a desired state in a Deployment, and the Deployment [Controller](https://kubernetes.io/docs/concepts/architecture/controller/) changes the actual state to the desired state at a controlled rate. You can define Deployments to create new ReplicaSets,